软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2016,27(5):1246−1265 [doi: 10.13328/j.cnki.jos.004828]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

一种适应 GPU 的混合 OLAP 查询处理模型*

张 宇[1,3], 张延松[1,2,3], 陈 红[1,3], 王 珊[1,3]

[1](数据工程与知识工程教育部重点实验室(中国人民大学),北京 100872)

[2](中国人民大学 中国调查与数据中心,北京 100872)

[3](中国人民大学 信息学院,北京 100872)

通讯作者: 张延松, E-mail: zhangys_ruc@hotmail.com

摘 要: 通用 GPU 因其强大的并行计算能力成为新兴的高性能计算平台,并逐渐成为近年来学术界在高性能数据库实现技术领域的研究热点.但当前 GPU 数据库领域的研究沿袭的是 ROLAP(relational OLAP)多维分析模型,研究主要集中在关系操作符在 GPU 平台上的算法实现和性能优化技术,以哈希连接的 GPU 并行算法研究为中心.GPU 拥有数千个并行计算单元,但其逻辑控制单元较少,相对于 CPU 具有更强的并行计算能力,但逻辑控制和复杂内存管理能力较弱,因此并不适合需要复杂数据结构和复杂内存管理机制的内存数据库查询处理算法直接移植到 GPU 平台.提出了面向 GPU 向量计算特性的混合 OLAP 多维分析模型 semi-MOLAP,将 MOLAP(multidimensional OLAP)模型的直接数组访问和计算特性与 ROLAP 模型的存储效率结合在一起,实现了一个基于完全数组结构的 GPU semi-MOLAP 多维分析模型,简化了 GPU 数据管理,降低了 GPU semi-MOLAP 算法复杂度,提高了 GPU semi-MOLAP 算法的代码执行率.同时,基于 GPU 和 CPU 计算的特点,将 semi-MOLAP 操作符拆分为 CPU 和 GPU 平台的协同计算,提高了 CPU 和 GPU 的利用率以及 OLAP 的查询整体性能.

关键词: GPU;联机分析处理;内存数据库;协同计算;数组计算

中图法分类号: TP311

中文引用格式: 张宇,张延松,陈红,王珊.一种适应 GPU 的混合 OLAP 查询处理模型.软件学报,2016,27(5): 1246−1265. http://www.jos.org.cn/1000-9825/4828.htm

英文引用格式: Zhang Y, Zhang YS, Chen H, Wang S. GPU adaptive hybrid OLAP query processing model. Ruan Jian Xue Bao/ Journal of Software, 2016,27(5):1246−1265 (in Chinese). http://www.jos.org.cn/1000-9825/4828.htm

GPU Adaptive Hybrid OLAP Query Processing Model

ZHANG Yu[1,3], ZHANG Yan-Song[1,2,3], CHEN Hong[1,3], WANG Shan[1,3]

[1](Key Laboratory of Data Engineering and Knowledge Engineering (Renmin University of China), Beijing 100872, China)

[2](National Survey Research Center, Renmin University of China, Beijing 100872, China)

[3](School of Information, Renmin University of China, Beijing 100872, China)

Abstract: The general purpose graphic computing units (GPGPUs) have become the new platform for high performance computing due to their massive parallel computing power, and in recent years more and more high performance database research has placed focus on GPU database development. However, today's GPU database researches commonly inherit ROLAP (relational OLAP) model, and mainly address how to realize relational operators in GPU platform and performance tuning, especially on GPU oriented parallel hash join algorithm. GPUs have higher parallel computing power than CPUs but less logical control and management capacity for complex data structure, therefore they are not adaptive for directly migrating the in-memory database query processing algorithms based on complex

* 基金项目: 中央高校基本科研业务费专项资金(16XNLQ0, 13XNLF01); 华为创新研究计划(HIRP 20140507, HIRP 20140510)

Foundation item: Basic Research Funds in Renmin University of China from the Central Government (16XNLQ0, 13XNLF01); Huawei Innovation Research Program (HIRP 20140507, HIRP 20140510)

收稿时间: 2014-04-08; 修改时间: 2014-06-10, 2014-12-01; 定稿时间: 2014-12-17

data structure and memory management. This paper proposes a GPU vectorized processing oriented hybrid OLAP model, semi-MOLAP, which combines direct array access and array computing of MOLAP with storage efficiency of ROLAP. The pure array oriented GPU semi-MOLAP model simplifies GPU data management, reduces complexity of GPU semi-MOLAP algorithms and improves their code efficiency. Meanwhile, the semi-MOLAP operators are divided into co-computing operators on CPU and GPU platforms to improve utilization of both CPUs and GPUs for higher query processing performance.

Key words: GPU; OLAP; in-memory database; co-computing; array computing

性能是数据库最重要的指标[1],也是数据库研究长期关注的目标.硬件技术的飞速发展对数据库性能提升具有重要推动作用,现阶段主要体现在两个方面:大容量内存开始取代传统的磁盘成为新的高性能数据存储设备;多核处理器及众核协处理器提供高达几百至几千的并行计算单元实现高并行计算.在新的硬件技术支持下,内存计算(in-memory computing,简称 IMC)成为企业级数据处理的主流技术[2,3].但单纯将数据存储从磁盘升级到内存受到新的“memory wall(内存墙)”[4]制约,还需要在多级缓存、多通道内存访问、多/众核并行计算等技术的共同支持下才能充分地发挥先进硬件的作用,以提升数据库的性能.因此,内存计算的高性能不仅仅取决于内存,还取决于现代多/众核处理器的强大并行计算能力,内存计算需要扩展为内存多/众核并行计算(in-memory multi-/many-core computing,简称 IMMC).以 NVIDIA GPGPU 和 Intel Phi Coprocessor 为代表的协处理器计算技术是当前众核并行计算的代表性技术,其强大的硬件级并行计算能力和不同于通用处理器的硬件架构使其成为新兴的高性能内存计算研究热点.当前,学术界的研究主要集中在 CPU/GPU 混合平台上的关系操作优化技术,尤其是 OLAP(on-line analytical processing)中计算代价较高的哈希连接在 GPU 上的优化技术,主要技术路线是根据 GPU 与 CPU 之间的 PCIe 通道数据传输性能、GPU 并行计算性能等相关因素,基于代价模型分析混合平台上的查询操作任务,创建分布式查询优化方案,提高整体 OLAP 查询处理性能.从关系操作符的实现算法层面上看,GPU 上的查询操作仍属于对传统操作符算法的一种“调优(GPU-conscious tuning)”技术,即在 GPU 端创建相应的内存数据结构,根据 GPU 硬件特性优化配置线程参数,根据 GPU 存储结构特性优化算法实现技术等.然而,传统关系操作模型是一种迭代处理模型,需要较多的分支指令,是面向数据密集型任务而优化设计的,并不适合于 GPU 计算密集型的性能优势.实际上,GPU 中虽有几百甚至几千的流处理器能够提供强大的向量计算能力,但对于复杂分支指令、迭代处理、线程间数据同步、大数据高延迟访问等操作的效率弱于通用处理器.客观地说,关系操作模型并不是适合 GPU 向量计算特征的理想的 OLAP 查询处理模型.

本文提出了一种以数组存储和向量计算为特点的 GPU semi-MOLAP 多维分析处理模型,将多维数据集用逻辑多维数组建模,创建虚拟的多维数据 CUBE,实现事实数据与各个维之间的多维数组地址映射;事实数据并不真正存储在巨大的多维数组地址空间中,而是采用关系存储方式将事实数据压缩存储,只存储虚拟多维数组地址空间中实际的事实数据,并附加以虚拟多维数组地址空间中各维的坐标,与 ROLAP(relational OLAP)模型中事实表采用外键和度量属性的存储方式相同,提高大数据存储效率.事实数据采用列数组存储,我们进一步将压缩的事实数据划分为多维索引(虚拟多维数组地址序列)和度量数据,多维索引计算是通过维坐标地址映射生成多维查询对应的虚拟事实数据 cuboid(子数据立方体),并生成度量数组位图或向量,用于标识其在度量属性数组中的下标位置的过程,实现基于虚拟 MOLAP(multidimensional OLAP)模式的直接多维数组访问.GPU semi-MOLAP 多维分析处理模型是面向 GPU 数组存储和向量计算特点而定制的 OLAP 查询处理模型,采用完全数组存储和数组计算方式实现多维查询处理,算法实现简单,不依赖复杂的数据结构,对于数据仓库应用中典型的星形和雪花形模型具有良好的适应性,同时在多核处理器平台也具有突出的性能.

本文第 1 节介绍 GPU OLAP 研究的相关工作.第 2 节给出本文提出的 GPU semi-MOLAP 多维分析模型的概念,实现技术和具体实现框架.第 3 节通过实验验证 GPU semi-MOLA 模型在 GPU 协同 OLAP 查询处理时的性能和有效性.第 4 节总结全文.

1 研究背景及相关工作

1.1 内存计算主要硬件平台

内存计算的性能主要体现在内存访问性能和并行计算性能.内存访问性能主要取决于内存带宽和内存访问延迟,内存带宽随着多通道内存访问技术的发展而提升,内存访问延迟由于物理电路的设计而难以得到大幅度提升,通常采用多级 cache 机制来降低频繁数据的访问延迟.并行计算性能随处理器核心数目及物理线程数目增加而增强.

如表 1 所示,当前的多核处理器已进入众核时代(通常高于 8 个内核的处理器称为众核处理器).

- 通用处理器,如 Xeon E7-4890 v2 有 15 个通用核心,其计算能力和逻辑控制能力均衡,线程较少,主频较高,支持的内存容量较大,但内存带宽较低;
- 至强协处理器(如 Phi 7120X)是一种专门面向高性能计算设计的 x86 架构众核(many integrated core,简称 MIC)处理器,内核数量和线程高于通用处理器但低于 GPU 流处理器数量,支持硬件级线程,采用类似通用处理器的缓存,并支持与 x86 兼容的编程模型;
- NVIDIA 公司的 GPGPU(如 Tesla K40)采用了众多计算核心和简单逻辑控制电路的硬件架构,具有强大的并行计算能力,但逻辑控制能力相对较弱,对于复杂数据结构和复杂指令的执行效率相对较低.

在内存访问性能上,通用处理器和 Phi 协处理器主要通过硬件级预取、多级缓存和多内存通道机制来降低数据访问延迟;GPU 则采用硬件级线程的零切换代价掩盖内存访问延迟的机制.在并行计算能力上,通用处理器和 Phi 协处理器都支持分支预测,但通用处理器支持乱序指令执行;Phi 协处理器支持顺序指令执行,并通过每个物理内核的 4 条硬件线程帮助屏蔽延迟对顺序指令执行应用可扩展性的影响.Phi 协处理器高性能的一个关键技术是 512 位寄存器和相应的 SIMD(single instruction multiple data,单指令多数据流)技术能够提供强大的向量数据处理能力;GPGPU 主要通过 SIMT(single instruction multiple threads,单指令多线程)技术更好地管理和执行多个单线程,避免 SIMD 机制中需要将数据组合为向量的代价,支持每个线程具有不同分支的执行代码.

Table 1 Hardware performance comparison of multicore CPU, Phi coprocessor and GPGPU

表 1 多核 CPU、Phi 协处理器、GPGPU 硬件性能对比

类型	Xeon E7-4890 v2	Xeon Phi 7120X	NVIDIA Tesla K40
核心数量/线程数量	15/30	61/244	2880 CUDA cores
主频	2.80 GHz	1.24 GHz	732MHz
内存容量	1536 GB	16GB	12GB
缓存容量	37.5MB	30.5MB	1.5MB
内存类型	DDR-3	GDDR5 ECC	GDDR5
内存带宽	85GB/s	352 GB/s	288 GB/s
价格	$6619.00	$4129.00	$5500.00

协处理器近年来较广泛地应于高性能计算领域,2013 年,使用 Phi 协处理器的天河二号夺得世界超级计算机 TOP 500 第一名[5],Phi 协处理器与 NVIDIA GPGPU 一同成为高性能计算的新平台.Phi 协处理器刚刚作为产品出现,新的型号不断推出,相关的研究还没有起步.而 GPGPU 的数据库应用研究起步较早,已积累了一些研究成果,并且 GPU 已广泛应用于高中低端计算平台,相对于协处理器计算平台需要增加额外的协处理器设备的前提,GPU 为计算机的标准配置,基于 GPGPU 的研究能够在不增加硬件成本的基础上提供额外的协同计算能力.

1.2 相关工作

1. MOLAP 和 ROLAP 模型在 GPU 中的应用

OLAP 领域研究长期受性能瓶颈的制约,近年来,GPU 由于其强大的并行计算性能而被研究人员所关注.传统 OLAP 实现技术包括基于多维数组存储的 MOLAP(multidimensional OLAP)和基于关系表存储的 ROLAP(relational OLAP)两种(还包括 HOLAP 模型,即细节数据保存在关系表中,聚合后的数据均保存在 CUBE 中).MOLAP 采用多维数组存储模型,稀疏的多维数据通过压缩划分为 chunk,多维查询的代价主要取决于数据扫描

和聚集计算代价.文献[6,7]在 GPU 上实现 MOLAP 中基于并行扫描和前缀求和原语的 GPU 聚集算法,开源 BI 软件 Palo 也提供了支持 GPU 的 Palo GPU Accelerator[8],通过多维数据的压缩和在多个 GPU 上的分布计算支持可扩展的 GPU 处理能力.文献[9]提出了一种基于协作关系的 CPU/GPU MOLAP 实现技术,CPU 和 GPU 分别用作数据处理和预处理平台,GPU 负责文本向数值的转换,CPU 负责 MOLAP 处理,通过调度机制协调两个平台上的任务.MOLAP 计算模型简单,更加适合 GPU 的高位宽 SIMT 并行计算.但 MOLAP 模型在稀疏数据存储效率、高维结构表示、维表动态更新等方面需要更高的存储代价及重构代价,在大数据 OLAP 中难以成为主流技术.

ROLAP 解决了稀疏多维数据存储效率问题,但 ROLAP 使用主-外键约束和基于值匹配的连接操作取代 MOLAP 中多维数据内在的多维地址映射,致使连接操作相对复杂.OLAP 多表连接的性能极大影响数据库整体性能,连接操作性能一直是数据库研究的关键问题,也是现代 BI 应用最大的障碍.基于 ROLAP 的内存数据库虽然相对于传统的磁盘数据库性能有了显著的提升,但内存计算的性能仍然是突出的瓶颈问题.

2. 面向 GPU 存储特点的数据分布优化策略

GPU 相对 CPU 的特点是拥有更多的并行计算单元和较少的逻辑控制单元,使得 GPU 适合于数值处理和高度并行的 SIMT 计算,但不适合复杂的逻辑控制和数据管理任务.因此,GPU 通常被用作数据库的“加速引擎”,相关研究包括在数据库管理模块中划分适合在 CPU 和 GPU 上处理的负载[10−14],提高 GPU 的高速显存利用率,最小化 PCIe 传输代价,通过计算和传输代价模型评估操作符在 CPU 和 GPU 上的处理代价,优化配置查询任务在异构计算平台上的执行等.GPU 显存容量远小于内存容量.对于计算密集型应用,最大 12GB 的高速显存能够满足一部分核心应用热点数据集显存本地(GPU memory resident)计算需求,在数据压缩技术[13,15]的支持下,GPU 显存的数据处理能力能够数倍甚至数十倍于 GPU 显存容量,GPU 的高并行计算性能可以得到充分的发挥,其计算效率相对于 CPU 有几十甚至数百倍的提升;对于数据密集型应用,需要处理的数据量远大于显存容量,PCIe 传输瓶颈取代传统数据库的 I/O 瓶颈成为 GPU 关系操作的新瓶颈,经过充分优化的 GPU 数据库性能通常仅为 CPU 的 2~4 倍,甚至可能低于 CPU 处理性能.对于数据库应用来说,需要根据模式和负载的特点划分出计算密集型的数据集和任务,根据 GPU 显存容量和负载优化配置硬件方案,实现 GPU memory resident 的数据分布和计算分配方案,在 CPU 端和 GPU 端同时实现内存计算,同时提高 CPU 与 GPU 的利用率,最小化数据传输代价.文献[16,17]将 GPU 作为数据仓库中较小的位图连接索引专用引擎来提高大数据的位图计算效率,提高了整体 OLAP 性能.现有研究中,主要将 GPU 内存用作 device 端的数据缓存,相当于在消除缓冲区机制的内存数据库中在 GPU 端重新增加一个 GPU 缓冲区.GPU 的数据缓冲机制主要通过 GPU 关系操作原语实现[11,12,18],还没有实现系统级的 GPU 缓冲区管理.内存数据库的核心是 cache-conscious 的优化技术,需要将 cache 容量和 cache 交换机制作为优化的基础;同样,GPU 数据库优化技术也需要将 GPU memory-conscious 作为核心思想,首先根据负载特点优化配置最佳的内存和 GPU 显存容量,然后再优化 GPU 数据处理时的显存利用效率.

3. 面向 GPU PCIe 通道性能瓶颈的优化技术

在当前的硬件技术水平下,PCIe 通道的性能成为制约 GPU 数据库性能最关键的因素.文献[19,20]揭示了混合 CPU/GPU 平台的性能远低于纯 GPU 平台的原因在于数据传输代价大.随着 PCIe 技术的发展和 2.0,3.0,4.0 标准的推出,GPU 的数据传输性能逐渐改善,但相对于 GPU 高速的显存带宽而言依然差距明显,PCIe 传输仍然是 GPU 的性能瓶颈.当前的相关研究主要体现在不同的传输优化技术,如 pinned 内存传输技术[13]、多通道内存传输(multiple memory channels)[14]、DMA(direct memory access,直接内存存取)访问以及数据压缩技术[21−24]等对提高 PCIe 通道性能的影响.但对于内存数据库而言,memory resident 的数据存储通常为 Pageable 存储(可分页面动态存储),对应较低的 PCIe 数据传输性能;而性能更高的 pinned(页锁定存储)内存数据访问机制在内存与 GPU 数据传输之前需要一个内存复制处理,增加了数据访问的预处理代价.与此相对,AMD 公司的 APU (accelerated processing unit,加速处理器)将处理器和显示器核心集成在一个芯片上,实现 CPU 与 GPU 的融合,共享相同的 L2/L3 cache 和内存控制器,通过 zero copy buffer[25]来消除 GPU 的 PCIe 通道传输代价,文献[26]在 AMD APU 上通过消除 PCIe 传输代价提升哈希连接性能.但相对于独立的 GPU,APU 中集成的核心数量极大地低于独立 GPU 的核心数量,并行计算能力有所降低.从 GPU 硬件发展趋势来看,显存容量持续增长,PCIe 带宽成

倍增加,计算核心数量不断提高,高性能独立 GPU 协处理器作为专用的查询加速引擎的技术格局可能仍然会持续下去,而 APU 可能会成为通用的加速引擎.

4. GPU“协处理引擎”优化策略

GPU 上一系列关系操作优化研究奠定了 GPU 作为数据库“协处理引擎”的实现基础.基于 GPU 的关系操作优化[12,27]验证了 GPU 上关系操作的实现与性能特点.文献[28]通过实验验证了 GPU 排序的性能相对于 CPU 的巨大性能优势.在 GPU 与数据库系统集成方面,文献[29]直接在 GPU 上实现了一个 SQLite 命令子集,从而将 GPU 操作原语集成到 SQL 命令集中,避免数据库开发者对 CUDA 编程的依赖.文献[13]通过 YSmart 进行 SQL 解析,生成 CUDA/OpenCL 后进行编译,并连接到一系列预置的 CUDA/OpenCL 查询操作符构成 SQL 查询处理执行计划.文献[30]针对数据仓库应用中大数据传输代价提出了 Kernel fusion 数据传输优化技术,通过编译框架 Kernel Weaver 自动融合关系操作符,以减少关系操作过程中冗余的数据移动代价,相当于一种 GPU 显存缓冲区交换机制优化思想.GPU 代价模型[10,13,25]是评估 GPU 处理性能的基础,文献[31,32]提出了一个基于统计方法的自调优决策查询代价评估模型,用于创建 CPU/GPU 混合查询执行计划.GPU 上的查询代价模型随着连接算法优化技术的不断改进而不断更新,难以精确地评估 GPU 执行代价.GPU 优化技术的核心技术与多核 CPU 类似,仍然是哈希连接优化技术,文献[33,34]中对哈希连接的优化技术被快速吸收到 GPU 优化技术研究中,文献[13,26]以优化的哈希连接算法对 GPU 优化性能进行深入研究和建模.文献[13]尤其研究了列存储数据库中 invisible join[35]算法在 GPU 中的性能,相对于大多数简单两表哈希连接算法研究,实现了面向完整 SSBM 基准的系统设计和性能测试.但 GPU 算法在 CPU 平台上执行的平均性能低于经典的内存数据库 MonetDB 性能,表明 GPU 上仍然缺乏真正适合 GPU 的高效 OLAP 算法,更多地是依赖 GPU 强大的硬件性能而获得整体性能提升.GPU 上的同步操作代价很高,文献[36]采用多级并行执行方式优化同步代价.Ocelot[37]集成在 MonetDB 中,通过 OpenCL 提供在不同硬件设备上的执行接口.Ocelot 在 GPU 显存数据集上有较好的性能,但在 TPC-H 测试中仍有部分查询性能低于多核并行 MonetDB.对于相同的算法,由于 GPU 在逻辑控制和复杂数据管理能力上弱于 CPU,OpenCL 在 CPU 上的算法性能往往低于优化的多核并行 CPU 算法,GPU 的综合性能对于深度优化的 CPU 算法优势并不明显,甚至在一些查询中,GPU 算法性能低于 CPU 算法.因此,GPU 对数据库的加速并不是全方位的,需要根据数据的特征和操作的特征组合 CPU 和 GPU 查询处理模块.

综上所述,随着 GPU 并行计算单元集成度迅速超过 CPU 以及新一代的并行编程框架 CUDA/OpenCL 的推动作用,通用 GPU 成为当前大数据实时分析处理的新平台.但受 GPU 硬件结构的制约,GPU 并不能完全优化数据库中多样的关系操作,对于依赖复杂内存结构,如链接哈希表、复杂迭代处理的关系操作来说,众多的研究表明,GPU 的性能优势并不突出,尤其是当前较低的 PCIe 带宽使 GPU 性能受到进一步的制约.因此,GPU 要不要用、怎么用、用在哪些处理过程,是新一代内存数据库系统所需要考虑的问题.

除了硬件性能约束之外,GPU 独特的编程框架也是 GPU 在数据库上应用的软件障碍,增加了数据库软件开发成本.我们不仅要考虑 GPU-conscious 的算法优化技术,同时也需要考虑 GPU-oblivious 的 GPU 数据库算法实现框架,提高 GPU 数据库技术对未来不同硬件平台的适应性.

2 GPU semi-MOLAP 多维分析模型

当前,GPU 上的 OLAP 研究主要集中在哈希连接在 GPU 上的优化技术,属于一种内存优化的算法针对 GPU 硬件特点的“调优”技术.GPU 的特点是向量计算能力强,复杂数据类型管理能力较弱,因此,基于内存动态链表结构的哈希连接算法并不是最适合 GPU 的算法.我们针对 GPU 的简单数据类型管理能力和强大的向量计算能力提出了一种基于数组存储和向量计算的 semi-MOLAP 模型,使其更加适合 GPU 的向量计算特点,并通过三阶段处理过程最小化 GPU 计算与 CPU 计算的耦合度,提高 GPU 计算平台的独立性和计算效率.

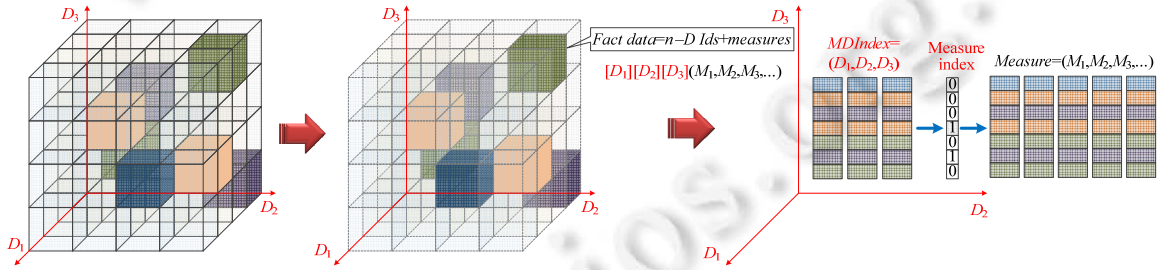
2.1 Semi-MOLAP模型概念描述

如图 1(a)所示,MOLAP 模型将多维数据集组织为空间立方体,每一个存储单元对应唯一的多维空间坐标,即多维数组下标,事实数据由多维数组下标代表其在多维数组中的物理位置.多维查询过程相当于根据查询条

件确定在各个维上的下标集合后,直接在多维数组空间中抽取满足查询条件的数据进行聚集计算的过程.MOLAP 多维查询是一个在庞大的多维数组空间中直接访问的过程,多维数组大小对 MOLAP 查询性能影响不大,查询性能主要取决于内存随机访问性能.在现实的企业级数据仓库应用中,数据立方体通常非常稀疏,如图 1(a)中 3 个维上 4×4×4 的多维空间中只有 7 个实际数据,数据存储效率非常低;而且数据立方体决定了数据的物理存储位置,一旦维发生改变,则需要对数据立方体进行重构,MOLAP 模型的更新代价非常高.

本文提出的 semi-MOLAP 模型如图 1(b)所示,多维数据存储采用虚拟数据立方体(virtual CUBE),即保留数据立方体的逻辑结构,将虚拟数据立方体中实际的事实数据进行序列化.即,将原有多维数组下标按维顺序序列化,将一个事实数据转换为 n 个维坐标 ID 和度量属性序列结构.虚拟数据立方体的序列化存储与 ROLAP 模型中事实表包含维表外键和度量属性的存储结构相类似,区别在于,ROLAP 模型中的事实表外键属性只需满足关系数据库的主-外键参照完整性约束条件即可;而虚拟数据立方体的序列化存储中维坐标 ID 代表了事实数据在虚拟数据立方体各维上的物理坐标位置,既符合 ROLAP 模型中主-外键参照完整性约束关系,又满足 MOLAP 模型中事实数据与各维度空间位置映射关系.虚拟数据立方体中的事实数据序列化没有特殊的要求,不需要参照多维坐标轴的顺序(如 Z-order),在实际应用中,首先需要将维表映射为虚拟数据立方体的多个维轴,事实数据遵循数据的物理存储顺序即可.

为了提高多维数据访问性能,我们将多个维坐标 ID 和度量属性存储为列数组,即,用数组存储各个事实数据属性,低势集的字符串类型属性采用轻量字典表压缩技术[22],数组中仅存储其较短的压缩编码.内存数据库[4]和 GPU 数据库[13]广泛使用数组作为物理存储数据结构,本文采用完全的数组物理存储技术将维表和事实表存储为数组族(array family),数组下标默认地用作数组间关联访问的索引.如图 1(c)所示,我们将事实数据进一步划分为两个部分:维坐标 ID 属性组和度量属性组.前者用作多维索引,负责在虚拟数据立方体上完成多维数据过滤,并生成基于位图或向量结构的度量索引(measure index);后者用于在度量数组中按度量索引指示的数组下标随机访问多维查询相关的数据.传统的基于 pipeline 的处理模式无法将 OLAP 负载中的计算密集型负载和数据密集型负载分离,因此,GPU 的高性能计算不得不受到数据传输延迟的消极影响.通过这种数据划分,多维查询可以分解为在较小的维坐标 ID 属性组上计算密集型多维过滤操作和在较大的度量属性组上数据密集型的聚集计算两个处理阶段,并使两个处理阶段分别适合于 GPU 较小内存上的高性能并行计算和 CPU 较大内存上的高性能随机数据访问,从而实现 GPU 与 CPU 在数据和计算上的协同处理,实现 GPU memory resident 和 CPU memory resident 计算,最小化数据传输代价.



(a) MOLAP 模型 (b) semi-MOLAP 模型 (c) semi-MOLAP 存储模型

Fig.1 MOLAP, semi-MOLAP and semi-MOLAP storage model

图 1 MOLAP,semi-MOLAP 和 semi-MOLAP 存储模型

在 MOLAP 模型中,多维查询可以转换为在各个维轴上的坐标,然后直接按多维坐标访问数据立方体中的查询相关数据.假设数据立方体对应多维数组 M[4][4][4],多维查询 Q 在 3 个维轴上对应的坐标分别为 3,2,3,则多维查询可以直接访问 M[3][2][3]完成查询处理任务.虚拟数据立方体并没有创建的物理多维存储空间,不能实现根据查询在各维轴上的坐标直接向事实数据的物理位置映射,但序列化的事实数据中存有其在各个维轴上

的坐标值,能够实现向各维轴坐标位置的映射.如图 2 所示,多维查询转换为在各维轴上的过滤条件并生成维过滤器,首先将虚拟数据立方体中的序列化事实数据在维 D_1 上进行过滤,得到一个切片(即满足维 D_1 上过滤条件的事实数据);然后,将该数据切片在维 D_2 上进行过滤,进一步缩减事实数据集;最后,通过维 D_3 上的过滤得到多维查询对应的数据集.

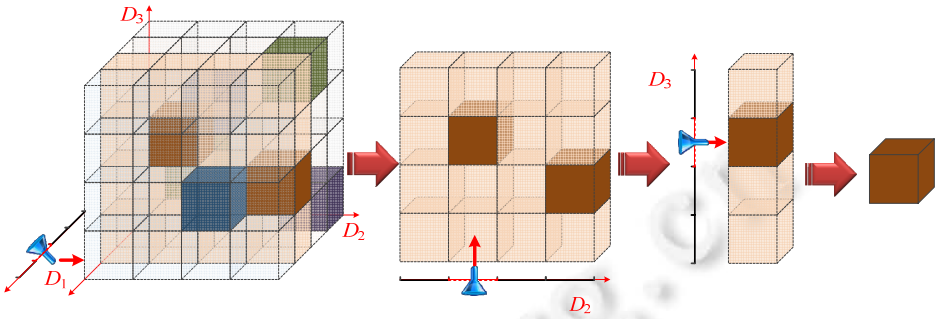


Fig.2 *n*-Dimension filtering

图 2 *n* 维过滤

在 semi-MOLAP 数组存储模型中,多维查询首先将谓词条件应用于相应各维并产生维过滤器(维过滤位图,将满足该维上过滤条件的维坐标位置标识为 1,不满足则标记为 0);维 ID 数组中存储的是事实数据对应的维坐标,通过将坐标值直接映射到维过滤器对应的下标位置完成维过滤操作.如图 3 所示,维 ID 数组(D_1)首先在维过滤器 $DFilter_1$ 上进行过滤,并将过滤结果记录在度量索引(measure index)中;然后,根据度量索引中指示的数组位置访问维 ID 数组 D_2,并在维过滤器 $DFilter_2$ 上进行过滤,同时更新度量索引中满足当前过滤条件的数组位置;当完成维 ID 数组(D_3)在维过滤器 $DFilter_3$ 上的过滤操作后,度量索引构造了满足当前多维查询条件的度量数据数组下标集合.由于多维查询的选择率非常低,根据度量索引按位置访问较大的度量数据和聚集计算可以高效地执行.

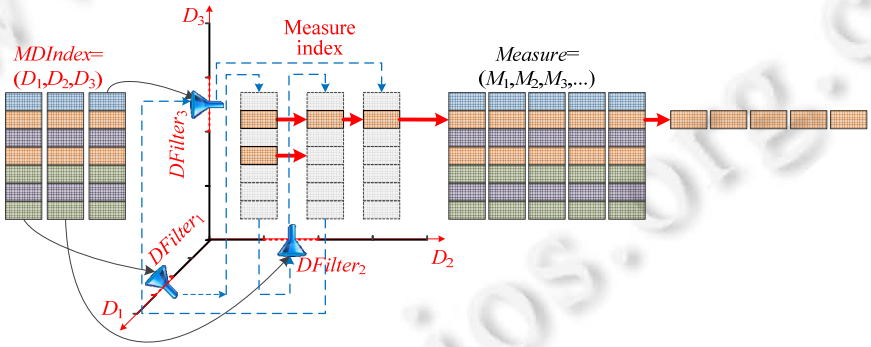


Fig.3 OLAP query processing model oriented to *n*-D filtering

图 3 基于 *n* 维过滤的 OLAP 查询处理模型

2.2 Semi-MOLAP模型实现技术

2.2.1 Semi-MOLAP 物理存储实现技术

MOLAP 模型以维属性为粒度组织多维数组集,如[*customer_gender*][*customer_education*][*customer_nation*][*product_catalog*][*supplier_region*][*year*]…(*Measure*$_1$,*Measure*$_2$,…),通常为高维数据集.ROLAP 模型则将相关维属性组织在维表中,维属性之间形成层次关系,如 SSB 中 date 维表的维属性 year-month-day 形成一个时间层次

结构.维表将 MOLAP 模型中的高维结构简化为星形或雪花形模型的低维结构,通过关系操作实现动态的高维查询处理,如在 SQL 命令使用 group by c_nation,c_gender,c_education 语句在维表 customer 中实现 3 个维度上的聚集计算.

Semi-MOLAP 模型采用与 ROLAP 模型兼容的存储模型,维表采用数组存储,数组下标用作隐式的维表主键,原始的维表主键退化为普通键码.相应地,事实表中的维表参照外键列需要在数据仓库的 ETL(数据抽取、加载、转换)过程中转换为相参照的维表记录数组下标,或者在数组存储的数据仓库中动态更新.semi-MOLAP 模型中的维成员(维表记录)采用原位更新(in-place update)机制,即,一旦分配了维成员,则保持在维上唯一的坐标位置,不支持传统数据库中先删除原始记录再插入新记录的异位更新(out-of-place update)机制.如图 4 所示,semi-MOLAP 模型中的 insert 命令对应维表数组单元的追加操作;update 命令在原始位置上直接进行属性值更新;delete 命令必须保证事实表中参照该维表记录的事实记录首先被删除的前提才能在维表中删除相应的维记录,维表设置一个删除向量(*DeletedVector*)记录维表中删除记录的数组下标;当维表插入新记录时,优先将删除向量对应的维表数组位置分配给新的维表记录.

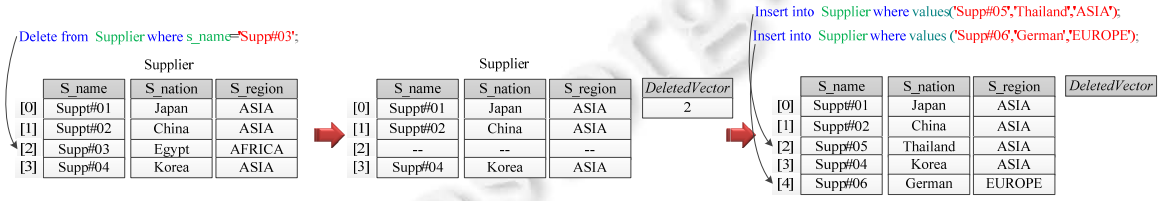


Fig.4 Update mechanism for semi-MOLAP model

图 4 semi-MOLAP 模型更新机制

semi-MOLAP 模型中,维表记录的增、删、改操作不影响已有的事实表数组存储结构,分析型内存数据库通常支持 insert-only 模式的更新操作,事实表只需要维护一个动态增长的数组结构即可,不需要额外的排序、索引等空间和时间代价大的预处理操作.

2.2.2 多维查询语言 MDQL(multidimensional query language)

semi-MOLAP 存储模型将 ROLAP 的星形或雪花形模型转换为多维数据模型,构建了事实表和维表之间的虚拟数据立方体,事实记录与维记录之间通过数组下标的参照引用建立了虚拟多维数组地址映射,不再需要查询命令中声明表间连接关系,多维查询命令可以简化为如下形式:

```
SELECT c_nation, s_nation, sum(l_revenue), sum(l_price)
FROM customer, lineorder, supplier
WHERE lo_custkey = c_custkey
and lo_suppkey = s_suppkey
and c_region='AMERICA'
and s_region='ASIA'
GROUP BY c_nation, s_nation;
```

```
OUTPUT: c_nation, s_nation, sum(l_revenue), sum(l_price)
FILTER:  c_region='AMERICA', GROUP: c_nation
FILTER:  s_region='ASIA', GROUP: s_nation;
```

其中,OUTPUT 语句用于定义查询结果数据立方体(基于分组的查询结果集)中的各个维度和聚集计算表达式;FILTE 语句定义了各个维表上的过滤条件;GROUP 语句定义了维上的分组属性集,可以为空集、单个分组属性或者多个分组属性集合.

多维查询语言 MDQL 的 n 个 FILTER 语句定义了一个虚拟的 n 维数据立方体,各维取值依据如下策略:

- 当维上没有分组属性时,维过滤器(*DFilter*)为一个位图(bitmap),用以标识每个维记录位置是否满足该维上的谓词条件(满足谓词条件的维过滤位图位置置为 1,否则为 0).
- 当维上有分组属性时,我们将满足 FILTER 条件的 GROUP 属性投影出来并建立一个数组字典表,并且以分组属性字典表数组下标(n 个成员的字典表数组下标映射为 0…n–1)作为值建立维过滤向量(满足谓词条件的维过滤向量位置取值为分组属性字典表数组下标,否则置为–1),维过滤向量预设了每一个

满足维过滤器的维表记录在多维查询结果数据立方体中当前维度上的坐标.

- 当维表上有多个 GROUP 属性时,我们将多个 GROUP 属性作为一个超级 GROUP 属性(super GROUP attribute)进行处理,维过滤向量中记录的值为该 GROUP 属性组在字典表数组中的下标.
- 当维上没有 FILTER 属性而只有 GROUP 属性时,该维不参与多维过滤操作,但 GROUP 语句对应的基于字典表压缩的属性为多维查询结果数据立方体提供一个聚集维度.

目前,MDQL 只支持标准 OLAP 中的 SPJGA 操作符(如全部 SSB 测试查询).对于复杂雪花形模型上的多维查询以及嵌套子查询等没有提供直接支持,我们将在未来的工作中将 MDQL 扩展到更加通用的 OLAP 应用场景中.

2.2.3 Semi-MOLAP 查询处理算法设计

Semi-MOLAP 查询处理可以分为 3 个处理阶段:(1) 构建虚拟数据立方体;(2) *n* 维过滤;(3) 聚集计算.

(1) 构建虚拟数据立方体

根据 DMQL 查询中 *n* 个 FILTER 的定义,动态创建虚拟的 *n* 维数据立方体中的 *n* 个维过滤向量,算法描述如下:

算法 1. 构建虚拟数据立方体.

```
INTPU:MDQL 命令 MDQLStatement.
OUTPUT:维过滤向量集合 DFilter.
BEGIN
  DFilter=∅;
  FOR MDQLStatement 中的每一个 FILTER 语句 F DO
    IF F.group IS NULL THEN
    BitDimFilter=new bitmap;   //如果没有 GROUP 属性,则创建维过滤位图
      BitDimFilter=genDFilter(F.filter);
      Add BitDimFilter to DFilter;
      BREAK;
    END IF
    IF F.filter IS NULL THEN
      DimFilter=new vector;   //如果没有 FILTER 属性,则创建 GROUP 属性维向量
      DimFilter=genDFilter(F.group);
      Add DimFilter to DFilter;
      BREAK;
    END IF
    DimFilter=new vector;
    DimFilter=genDFilter(F.filter,F.group);   //根据 FILTER 和 GROUP 语句创建维过滤向量
    Add DimFilter to DFilter;
  END FOR;
  Sort(DFilter);   //对维过滤向量集合按向量选择率排序,低选择率维过滤向量优先使用
  Return DFilter;
END.
```

算法返回按选择率升序排列的维过滤向量集合,用以构建虚拟数据立方体的 *n* 个维轴.

以后文图 7 为例,查询只涉及两个维表,在构建虚拟立方体阶段需要创建两个维轴作为过滤向量.首先,根据维表上的过滤条件投影出相应的分组属性(或分组属性组);然后,为其建立分组属性字典表并将其字典编码写入维过滤向量对应的位置.生成维过滤向量的时间复杂度为 $O(n)$,向量为与维表等长的 *k* 位字典编码,空间复杂

度为 $O(n)$.由于维表记录数量较少,因此处理时间非常短.

(2) n 维过滤

在构建虚拟数据立方体阶段,只需要将 SQL 命令转换为 MDQL 命令,并通过对维表的处理生成多维查询所需要的 n 个维过滤向量(按选择率升序排序),不涉及事实表上的数据处理,因此可以使用与事实表不同的查询处理引擎,例如功能完整的内存数据库系统 MonetDB 或 Vectorwise 等来支持复杂的维表数据类型、空值处理以及谓词表达式等,不需要重新开发一个功能完整的内存维表数据库,只需要嵌入式的维表查询处理引擎提供标准的维过滤向量集即可.

维过滤向量集确定了事实数据中 n 个维 ID 数组的扫描顺序,最终将满足所有维过滤器的维 ID 数组下标作为度量索引,算法描述如下:

算法 2. n 维过滤.

INPUT:维过滤向量集合 *DFilter*,维 ID 数据集合 *DimIDs*;

OUTPUT:度量索引 *MeasureIndex*.

```
BEGIN
  Initiate(MeasureIndex);   //初始化度量索引,预置初值为 0
  FOR DFilter 中的每个维过滤向量 DimFilter DO
    DimID=getAddressOf(DimFilter,DimIDs);   //获得当前维过滤向量对应的维 ID 数组地址
    Filtering(DimFilter,DimID,MeasureIndex)→MeasureIndex;
    //以当前度量索引为基础对维 ID 数组按位置随机访问并进行维过滤操作,更新度量索引
  END FOR;
  Return MeasureIndex;
END.
```

如后文图 7 示例:在维过滤向量构建阶段,已经按查询中的 GROUP 属性预构建了查询结果数据立方体各维,并将维 ID 记录在维过滤向量中作为分组聚集多维数组在各个维上的坐标分量.在 n 维过滤算法中,对度量索引 *MeasureIndex* 的更新过程是一个迭代计算查询结果数据立方体多维坐标的过程.算法返回经过 n 个虚拟维过滤后所产生的度量索引,用于对度量属性按数组地址直接访问,度量属性根据度量索引中的多维坐标值确定其聚集计算在所依赖的多维数组(查询结果数据立方体)中的下标.多维过滤是一个逐级按选择率递减的过程,当选择率较低时,维数的增多对多维过滤的性能影响不大,多维过滤处理的时间复杂度为 $O(n)+O(s_1n)+O(s_1s_2n)+\ldots$,其中,$s_1,s_2,\ldots$为各维上的选择率.$n$ 维过滤阶段复用度量索引,在每维过滤中迭代更新度量索引,不需要额外的存储空间,时间复杂度为 $O(n)$.

(3) 聚集计算

根据 MDQL 命令中 OUTPUT 语句确定聚集计算的数据立方体结构,并根据 n 维过滤所生成的度量索引 *MeasureIndex* 访问相应的度量属性值,实现基于数据立方体的聚集计算,算法描述如下:

算法 3. 基于数据立方体的聚集计算.

INPUT:MDQL 命令 *MDQLStatement*,度量索引 *MeasureIndex*,度量属性集 *MeasureAttr*;

OUTPUT:多维查询结果集 *ResultSet*.

```
BEGIN
  Initiate(MDQLStatement.output,MeasureIndex)→ResultCUBE;
  //根据 MDQL 命令中的 OUTPUT 语句和度量索引中的多维数组下标构建查询结果数据立方体
  Agg(MeasureAttr,MeasureIndex)→ResultCUBE;
//根据度量索引按下标访问度量属性数组,并将聚集表达式结果写入度量索引值对应的查询结果数据立方体对应的数组下标位置进行聚集计算
  Decoding(ResultCUBE)→ResultSet;
```

//将多维查询结果数据立方体通过维向量字典数组还原为原始分组属性结果集

Return *ResultSet*;

END.

度量索引中记录了满足最终连接条件的事实记录的位置和其在多维分组聚集数组中的下标,可以实现对事实数据高效率的随机访问和直接聚集,该处理过程的时间复杂度为 $O(s_1*s_2*\ldots*n)$.聚集计算过程中使用预创建的多维查询结果数据立方体作为聚集器,由于 OLAP 查询中通常为低势集分组属性,其空间复杂度可以近似为 $O(1)$.

semi-MOLAP 对应的 3 个查询处理阶段彼此独立,可以将不同查询的 3 个处理阶段流水并行处理,也可以将它们分派到不同的处理器平台,如 CPU 和 GPU 平台协同处理.

2.3 GPU semi-MOLAP模型实现技术

2.3.1 数据分布策略和协同式 OLAP 查询处理策略

OLAP 多维数据集的维表和事实表是一种数据量偏斜的结构,其中,维表数量所占的比重非常低,数据集中比重最大的是度量属性.现实应用中,事实表度量属性的数量可能多达几十至上百个.事实表外键相当于度量属性的多维索引,数量和数据量相对较低.在星形测试基准 Star Schema Benchmark(SSB)中,当 *ScaleFactor*=20 时,4 个维表、事实表的 4 个外键及事实表度量属性所占比重如图 5 所示.

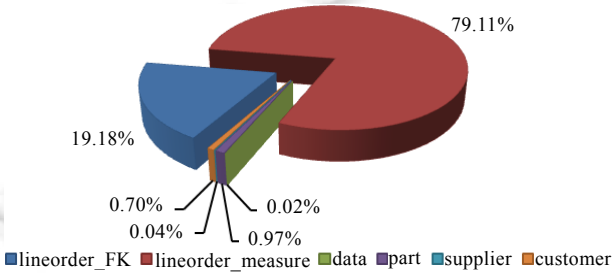


Fig.5 Data distribution in star schema benchmark (*ScaleFactor*=20)

图 5 SSB 中的数据分布(*ScaleFactor*=20)

在 CPU/GPU 混合计算平台上,内存容量较大,作为 OLAP 数据集的持久存储设备,GPU 显存通常为几个 GB 至十几个 GB.从 OLAP 数据集的数据分布特征和内存及 GPU 显存容量特点来看,最佳的数据分布方案是将 OLAP 数据集中数据量较小、计算密集型、计算的输入/输出数据量小的数据子集驻留于 GPU 内存、最小化内存与 GPU 之间低带宽 PCIe 数据传输的代价.从 OLAP 数据集来看,维表能够满足 GPU 显存驻留需求,但现实应用中,维表更新较频繁,而且数据类型多样,其上的操作多为扫描操作,并不适合 GPU 存储和计算;度量属性数据量大,多维查询极低的选择率,使其并不适合 GPU 存储;事实表外键在 semi-MOLAP 模型中用作多维索引,在 n 维过滤操作中涉及 PCIe 通道传输的数据包括极小的维过滤器和所生成较小的度量索引,能够最小化数据传输代价.

我们确定了事实表外键驻留 GPU 内存的数据分布方案后,一方面可以根据 GPU 内存的大小反向推导能够支持的最大 SSB 数据集大小,确定支持 CPU/GPU 最佳计算模式的数据集大小,根据 GPU 硬件条件配置最优的 CPU 计算平台;另一方面,当需要处理的数据集中事实表外键超过 GPU 内存时,我们对事实表外键进行水平分片,划分出 GPU 内存计算的数据分片和 CPU 内存计算的数据分片,由 CPU 和 GPU 协同完成 semi-MOLAP 的 n 维过滤操作,支持可扩展数据集上的协同计算,并由 CPU 完成两个度量索引上的度量聚集计算.semi-MOLAP 模型 3 个处理阶段的松耦合结构和基于较小数据交换的机制保证了 CPU,GPU 协同计算平台的效率.如图 6 所示,在一个中档服务器上,CPU 与 GPU 的计算性能相近,semi-MOLAP 以 GPU memory resident 为核心的数据分布策略能够保证 GPU 尽可能地发挥其并行计算性能、最小化数据传输延迟,并且充分发挥 CPU 和 GPU 的协同

计算能力,即使面对 GPU 内存无法支持的大数据处理任务,也能让 CPU/GPU 协同计算平台最大化 CPU 和 GPU 的利用率.

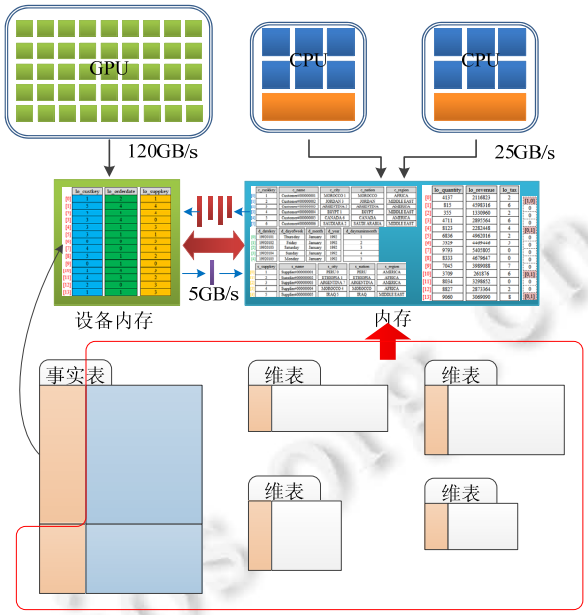


Fig.6 Co-Computing between CPU and GPU

图 6 CPU 和 GPU 上的协同计算

2.3.2 算法执行框架

在系统初始化阶段,按 GPU 内存大小将存储于内存中事实表外键的全部或者最大水平分片复制到 GPU 内存,构建多维索引加速器.图 7 为第 2.2.3 节算法执行示例.

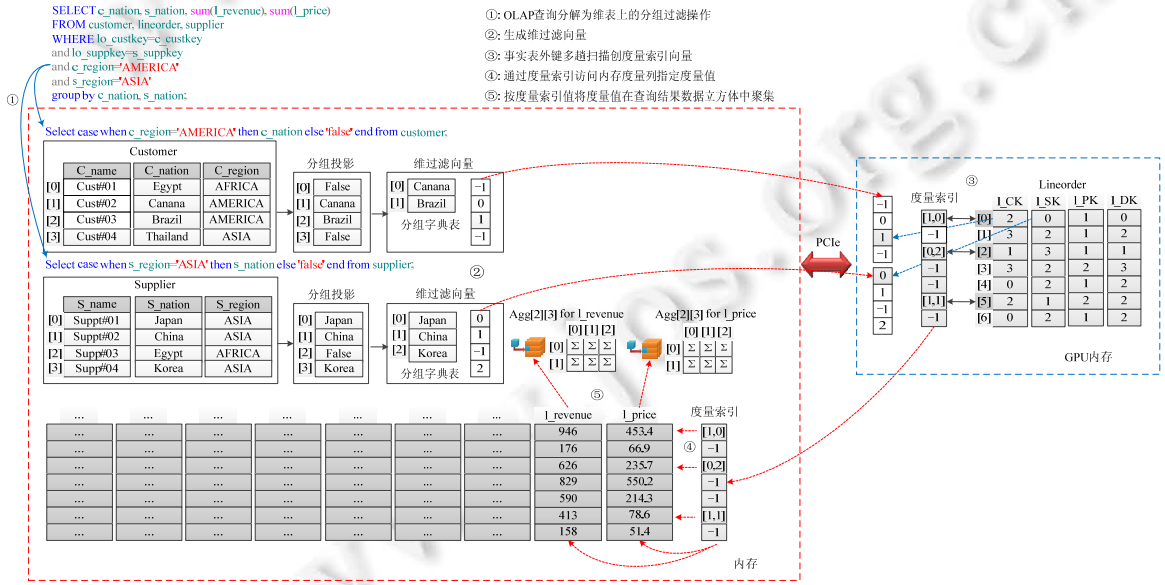


Fig.7 GPU semi-MOLAP execution framework

图 7 GPU semi-MOLAP 执行框架

SQL 命令中包含 customer 和 supplier 两个维表上的分组过滤操作,SQL 命令转换为 MDQL 命令后,由 CPU 首先构建虚拟数据立方体的两个维轴,生成两个维过滤向量,并通过 PCIe 通道传输到 GPU 内存;GPU 根据维过滤向量执行 GPU 端的 *n* 维过滤操作,完成基于 CUDA kernel 函数的并行多维索引计算,生成度量索引向量;度量索引向量通过 PCIe 通道传输回内存,由 CPU 完成对度量属性的随机访问,完成根据维过滤向量构建出的查询结果数据立方体上的聚集计算;最后,通过维过滤向量生成阶段创建的分组字典表对查询结果数据立方体解码,还原为标准的查询输出结果.

在 GPU semi-MOLAP 执行框架设计中,通过最大化利用 GPU 内存的策略,GPU 并行计算资源被最大化利用.对于多个查询并发处理过程,GPU 上操作是完全串行执行的.在 SQL 命令执行的 3 个阶段,两个执行时间较短的维表处理和度量计算阶段可以分解为独立的查询子任务,由 CPU 异步完成.不同查询之间的 3 个执行阶段在 CPU 和 GPU 处理器上可以流水并行.图 8 显示了 GPU semi-MOLAP 的流水执行框架,查询 *Q*1 在 GPU 执行时 *Q*2 的维表处理和 *Q*1 的度量计算阶段可以流水并行地在 CPU 上执行.semi-MOLAP 将 MDQL 查询分解为 3 个独立的处理过程,每个过程只依赖于独立的向量数据结构,CPU 与 GPU 平台上的计算采用的是异步执行方式,最小化 GPU 的等待时间.

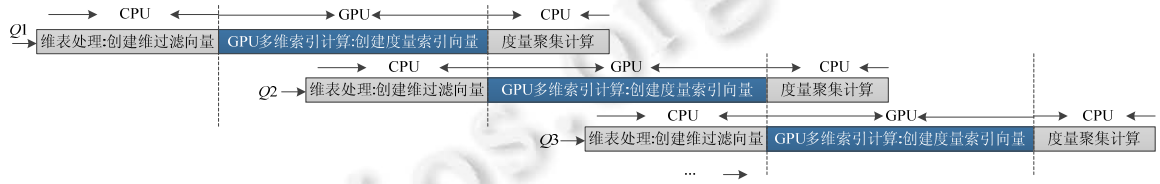


Fig.8 GPU semi-MOLAP pipelining execution framework

图 8 GPU semi-MOLAP 流水执行框架

综上所述,GPU semi-MOLAP 是一个松耦合的基于数据交换的执行框架,维表处理阶段可以采用已有的内存数据库作为维表查询处理引擎,负责复杂数据类型的存储、压缩、更新等数据库通用功能,不需要完全自主开发,只需要提供输出维过滤向量的 API 即可,从而降低了系统开发成本.GPU 多维索引计算和度量聚集计算是完全基于数值型数组的计算过程,数据管理和计算模型简单,易于进行多核 CPU 和 GPU 平台上的并行计算,也易于将核心的 GPU semi-MOLAP 算法集成到已有的内存数据库中.semi-MOLAP 模型同样适合 Intel Phi 协处理器计算平台,是一种高可扩展的众核高并行算法.

3 实 验

3.1 实验平台及Benchmark

本文实验平台为一台 Lenovo 工作站 ThinkStation D30,硬件配置为 2 个 Intel E5-2667 六核处理器,每处理器 L3 缓存容量 15MB,12GB DRAM,配置有一块 NVIDIA Quadro 5000 GPU,352 个流处理器,2.5GB GDDR5 显存,显存位宽为 320 位.操作系统为 ubuntu-12.04(precise)64-bit,kernel Linux 3.8.0-29-generic,CUDA 版本 5.5.相对于很多文献中的桌面级多核 CPU,本文实验平台的两块至强处理器具有良好的性能,Quadro 5000 GPU 虽然不是专业计算的 GPGPU,但代表了中端服务器配置的一般规律,即较强的多路 CPU 和中端 GPU.本文研究的意义并不在于评估哪个计算平台性能更好,而是在既有的通用中端服务器配置下更好地挖掘已有的中端 GPU 的计算能力,扩展服务器的计算能力.

3.1.1 测试数据集和数据分布策略

实验采用 SSB 数据集,数据集大小见表 2.其中,4 个维表过滤向量总大小为 1.63MB(每向量 8 位宽,每个聚集维度上支持 2^8 个成员),GPU 中需要驻留存储 4 个事实表外键列和一个度量索引向量,共计 2.24GB,占 GPU 显存容量的 87%.也就是说,当前 Quadro 5000 2.5GB 显存能够支持 *SF*=20 的 SSB 数据集的完全 GPU 多维索引计

算.当需要处理的数据量增长时,一方面可以按比例增加 GPU 数量,另一方面可以以 GPU 显存处理容量为基准对事实表外键属性组水平划分,由 CPU 和 GPU 协同完成多维索引计算,由 CPU 完成全部度量聚集计算.

Table 2 SSB dataset distribution (*SF*=20)

表 2 SSB 数据集分布(*SF*=20)

	lineorder_FK	lineorder_measure	date	part	supplier	customer
rows	120 000 000		2 555	1 064 386	40 000	600 000
Tuple width (byte)	82		90	91	106	116
	16	66				
Table size (GB)	1.79	7.38 (total 9.16)	0.000 2	0.090	0.004	0.065
Vector size (MB)			0.002	1.015	0.038	0.572
%	98.29		0.002	0.97	0.04	0.70
	19.18	79.11				

3.1.2 系统实现技术

GPU semi-MOLAP 采用 CUDA/C++语言开发,采用完全数组模型,使用 C++ vector 模板类支持动态数组上的 insert-only 更新,维表和事实表存储为数组族(array family),维表采用字典表压缩,维表以数组下标为主键,事实表外键用作多维索引,在 ETL 过程转换为相应维表数组下标.事实表外键属性采用 pinned 内存数组,预置与维表等长的维过滤向量和度量索引向量,事实表外键与度量索引在系统初始化时加载到 GPU 内存驻留,在查询处理过程中,每个查询对应的维过滤向量和度量索引在内存和 GPU 内存之间进行数据交换.基于 pinned 内存分配机制下维过滤向量在 PCIe 通道上的传输速度达到 6GB/s,4 个定长维向量的传输时间为 0.23ms,度量索引向量从 GPU 向内存的传输时间为 20.89ms,由于每个查询对应的维过滤向量和度量索引向量为定长,GPU 数据传输代价相对恒定.

3.2 实验结果及性能分析

3.2.1 实验对比结果

我们使用 MonetDB(Feb2013-SP5)与 GPU semi-MOLAP 在 SSB 上进行性能对比,如图 9 所示.我们通过 Dbvisualizer8.0.9 在 MonetDB 上执行 13 个测试查询,每个测试查询连续执行 5 遍,取最稳定的平均时间作为查询执行时间.与文献[13]不同的是,GPU semi-MOLAP 算法在多核 CPU 平台上性能全面超过 MonetDB,GPU semi-MOLAP 框架虽然面向 CPU/GPU 协同平台进行定制,但在多核 CPU 平台同样有较优的性能.

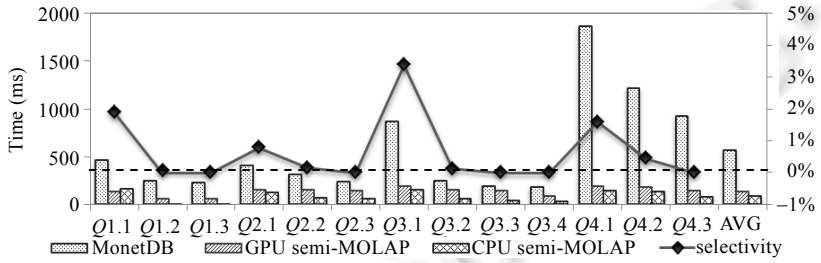


Fig.9 GPU semi-MOLAP, CPU semi-MOLAP and MonetDB performance comparison

图 9 GPU semi-MOLAP,CPU semi-MOLAP 和 MonetDB 性能对比

从图 9 中我们可以看到,CPU semi-MOLAP 总体性能(平均查询执行时间)优于 GPU semi-MOLAP(为查看方便,选择率坐标轴向上平移一个单位).MonetDB 每组查询执行时间与选择率成正向关系,选择率越高,连接操作和聚集计算代价越大,整体执行时间越长;各组之间查询执行时间主要受连接操作的复杂度影响,连接操作越多,查询执行时间越长,例如 *Q*1.1 选择率为 1.9%,*Q*2.1 选择率为 0.8%,*Q*3.1 选择率为 3.4%,*Q*4.1 选择率为 1.6%;但执行时间主要是按连接维表的数量排序的,如 *Q*4.1 选择率居于第三,但查询执行时间最长.与 MonetDB 相对,semi-MOLAP 算法执行时间基本上由选择率决定,连接表的数量对查询整体执行时间有一定的影响,但不如

MonetDB 那样显著,主要原因是 semi-MOLAP 算法采用 *n*-维过滤技术,每一次连接都极大地缩减下一个连接的维 ID 列中候选连接集大小,虽然维过滤的数量线性增加,但维过滤所操作的数据却呈指数级减少.也就是说,semi-MOLAP 算法更加适合高维查询.

图 10 对比了 GPU semi-MOLAP 算法和 CPU semi-MOLAP 算法的平均执行时间分解.GPU semi-MOLAP 算法是 CPU 和 GPU 平台协同计算技术,也就是说,GPU 只完成 GPU 内存中的 *n*-维过滤计算,生成度量索引并由 CPU 完成聚集计算.维表处理过程由 CPU 完成,维向量由于数据量极小(当 *SF*=20 时为 1.63MB),PCIe 传输延迟极低.GPU 和 CPU 算法性能差异主要体现在 CPU 和 GPU 的多维过滤操作和度量索引由 GPU 传输到 CPU 的延迟.度量属性上的聚集计算涉及较大的数据集由于极低的选择率在 CPU 上的计算效率极高,因此我们不考虑 GPU 上的传输代价较大的聚集计算,而是以 CPU 作为专门的聚集计算平台.

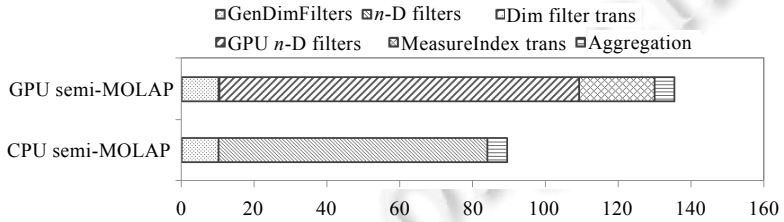


Fig.10 Breakdown of average query processing time for GPU semi-MOLAP, CPU semi-MOLAP

图 10 GPU semi-MOLAP 和 CPU semi-MOLAP 平均查询时间分解

表 3 详细列出了 GPU semi-MOLAP 算法和 CPU semi-MOLAP 算法各个执行阶段的时间.MonetDB 虽然与 semi-MOLAP 算法同样采用数组列存储和数组计算,但 MonetDB 仍然是一种 ROLAP 引擎,多维查询采用的是列存储多表连接技术,与 MOLAP 的数组地址直接映射技术相比,哈希连接和哈希分组聚集等操作需要更多的 CPU cycle 才能完成 MOLAP 对应的数组地址映射,整体性能低于 semi-MOLAP 算法.semi-MOLAP 算法是一种混合 ROLAP 与 MOLAP 的多维计算模型,多维查询处理性能介于理想的 MOLAP 模型和 ROLAP 模型之间.本文实验平台 GPU 显存容量只能支持 *SF*=20 的数据集的 GPU memory resident 计算,在多查询流水处理模式下可以支持 *SF*=40 的数据集平均分布在 GPU 和 CPU 平台,由 CPU 和 GPU 同时执行各自维表 ID 列上的多维过滤计算,再由 CPU 完成全局聚集计算,在 *SF*=20 数据集的 GPU 计算时间内完成整体 *SF*=40 的 CPU/GPU 协同计算.

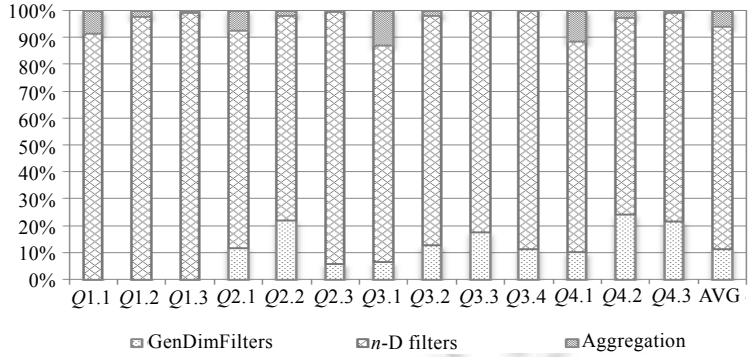
Table 3 SSB query execution time details (*SF*=20)

表 3 SSB 查询执行时间明细(*SF*=20)

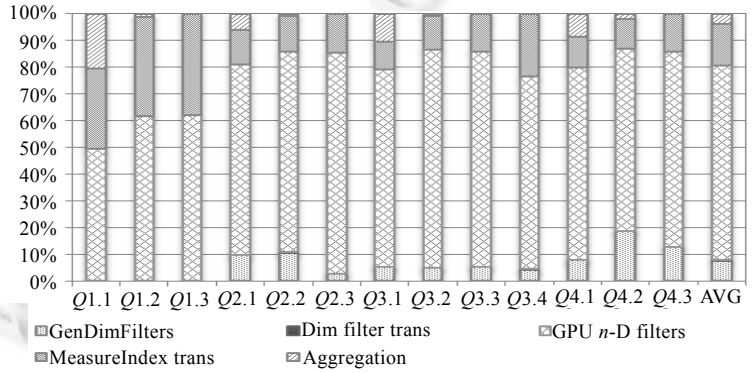
Queries (ms)	MonetDB	GenDim filters	*n*-D filtering	Aggregation	CPU semi-MOLAP	Dim filter trans	GPU *n*-D filtering	MeasureIndex trans	GPU semi-MOLAP
Q1.1	470	0.04	155.95	14.21	167.32	0.04	34.21	20.96	135.58
Q1.2	247	0.01	26.47	0.63	26.93	0.04	34.36	20.89	60.16
Q1.3	234	0.01	26.58	0.17	26.37	0.04	34.32	20.89	58.83
Q2.1	413	15.45	104.32	9.62	129.39	0.25	113.74	20.66	159.73
Q2.2	317	16.67	57.70	1.42	75.78	0.25	117.75	20.66	156.75
Q2.3	243	3.62	56.39	0.22	60.24	0.25	117.71	20.66	142.46
Q3.1	870	10.21	126.03	20.24	156.47	0.21	141.72	20.49	192.86
Q3.2	246	7.75	51.97	1.18	60.90	0.21	130.21	20.49	159.84
Q3.3	198	7.74	35.93	0.08	43.74	0.21	119.38	21.20	148.61
Q3.4	180	3.97	30.87	0.00	34.83	0.21	67.26	21.97	93.41
Q4.1	1866	15.39	118.62	17.29	151.29	0.39	140.53	21.85	195.45
Q4.2	1221	33.91	102.03	3.67	139.61	0.42	125.11	20.45	183.56
Q4.3	929	18.42	66.00	0.51	84.93	0.42	106.91	20.47	146.74
AVG	571.87	10.24	73.49	5.33	89.06	0.23	98.71	20.89	135.40

图 11 分别显示了 CPU semi-MOLAP 和 GPU semi-MOLAP 执行时间分解比例图.在 CPU semi-MOLAP 算法执行过程中,聚集计算的时间主要由选择率决定,我们在算法中采用基于多维数组的查询结果数据立方体技

术,每一个度量属性值可以根据度量索引中记录的数组下标直接定位多维数组单元,而且多维数组存储效率高,聚集计算的性能几乎不受查询结果中记录数量的影响.在 GPU semi-MOLAP 算法执行过程中,*n*-维过滤计算是查询执行时间的主要组成部分,度量索引列的 PCIe 传输代价所占的比例也较高.PCIe 传输延迟会随着 PCIe 通道标准的提升而成倍减少(如 PCIe3.0 带宽达到 16GB/s,而 PCIe2.0 为 8GB/s).



(a) CPU semi-MOLAP



(b) GPU semi-MOLAP

Fig.11 Breakdown of query processing time proportion for CPU semi-MOLAP and GPU semi-MOLAP

图 11 CPU semi-MOLAP 和 GPU semi-MOLAP 执行时间百分比分解

图 12 对比了 CPU 和 GPU 上的 *n*-维过滤计算性能.

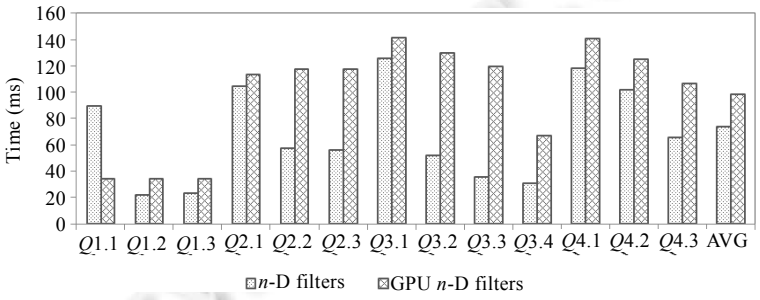


Fig.12 *n*-D filtering performance comparison on CPU and GPU platform

图 12 CPU 和 GPU 平台上的 *n* 维过滤操作性能对比

CPU 平台上采用 vector 存储压缩的度量索引,即,只存储满足当前维过滤条件的度量属性值的数组下标,每

次维过滤操作需要更新 vector 向量,将不满足当前过滤条件的向量值用其后满足当前过滤条件的向量值覆盖,完成这一过程需要较复杂的逻辑控制语句.而在 GPU 平台上我们采用的是向量计算技术,即度量索引是一个与维 ID 数组等长的数组结构,维过滤计算操作需要读取当前度量索引数组值,判断是否为–1:若不是,则进行当前维上的过滤操作,并将过滤操作结果(若不满足当前过滤条件则置为–1,否则更新为当前维上的查询结果数据立方体的迭代计算数组下标)写回原始度量索引数组位置.通过这种向量处理技术,最大化 GPU 的 SIMD/SIMT 并行计算能力,减少复杂逻辑控制语句的使用,提高 GPU 代码效率.但即使是当前最高的 512 位 SIMD 计算能力也只能一次处理 16 个 int 数据(32 位),对于 SSB 最高选择率为 3.4%,最低仅为 0.000076%,CPU 上基于紧凑数组下标的过滤访问效率高于 GPU 上的向量计算效率.

Semi-MOLAP 模型在 SSB 数据集上 4 个维过滤向量总大小为 1.63MB,远小于 E5-2667 处理器 15MB L3 cache 的大小,在 *n*-维过滤计算中具有良好的 cache 局部性.但在 GPU 平台上,只有 date 维过滤向量小于 32KB 的共享内存容量,能够实现共享内存数据访问,其他维向量不得不采用内存访问.因此,*n*-维过滤计算产生维过滤向量和维 ID 数组两种数据的内存访问延迟,在数据访问局部性方面性能不如 CPU 的多级缓存结构.

3.2.2 与相关工作性能对比分析

相关的研究[13,26,37]给出了 GPU 在 SSB,TPC-H 以及哈希连接实验中优于 CPU 的性能.一方面,这些研究中 CPU 的配置相对于 GPU 较低(一块四核 CPU,但 GPU 配置高于本文所使用的 Quadro 5000),因此在 CPU 和 GPU 两个平台的计算能力上并不对等;另一方面,这些研究采用传统的哈希连接算法,查询处理过程中的计算代价较大,GPU 能够更好地发挥强大的并行计算能力.

为了适合 GPU 数据处理特性,GPU 上的 OLAP 算法效率通常低于 CPU 算法效率,需要依赖 GPU 强大的硬件并行处理能力弥补算法效率的不足.而 semi-MOLAP 算法在 CPU 平台上同样具有较高的算法效率,图 13 为 CPU semi-MOLAP 算法与内存数据库 MonetDB 在不同线程下的并行加速比(*SF*=100,8 核 CPU×4),CPU semi-MOLAP 算法在 CPU 平台具有更好的加速比性能.因此,本文提出的 semi-MOLAP 计算模型的性能一方面体现在 semi-MOLAP 算法更好的 OLAP 处理性能和效率,另一方面体现在 semi-MOLAP 计算模型对数据和计算按 CPU 和 GPU 特点进行的分布处理策略,充分发挥不同计算平台的特点,提高 OLAP 的综合性能.

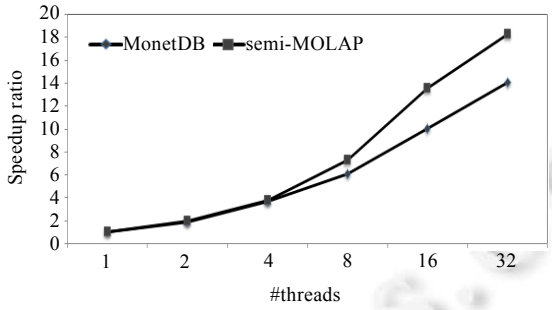


Fig.13 Speedup ratio of semi-MOLAP algorithm on CPU platform

图 13 semi-MOLAP 算法在 CPU 平台上的加速比

我们在实验中的平台配置有两个 E5-2667 六核处理器,共 12 个计算核心(24 个物理线程),而 Quadro 5000 的处理器(multiprocessors)数量为 11 个(共 352 个计算单元),数据处理单元数量对等,计算能力处于对等的水平.即使不考虑 CPU 和 GPU 之间的数据传输延迟,CPU 和 GPU 内存计算的性能处于相近的水平.从算法效率上看,semi-MOLAP 模型是一种多维数组访问技术,多维查询处理时的计算代价极低,主要为数据访问代价,性能由内存数据访问延迟、cache 容量、随机访问性能等因素决定.CPU semi-MOLAP 计算受益于较大的多级缓存机制和高效率的低选择率随机访问算法设计,在数据访问性能方面,CPU 优于 GPU,因此,本文得出了在对等计算能力的 CPU 和 GPU 平台上 CPU 略优于 GPU 协同计算性能的结论.

4 结束语

随着NVIDIA GPGPU,AMD APU和Intel Phi协处理器技术的迅速发展和成熟,协处理器计算已经成为高性能计算的新平台.协处理器平台与通用处理器平台在硬件特性和程序设计上存在较大的差异,将多核处理器优化的算法迁移到协处理器平台并面向协处理器硬件特性进行优化,并不能充分地发挥协处理器的硬件性能.本文首先从OLAP多维查询模型出发,提出了一种ROLAP和MOLAP相结合的多维查询模型semi-MOLAP,通过虚拟数据立方体建立了数组存储和数组计算模型,消除 ROLAP 中计算复杂度较高的关系操作符;同时,采用ROLAP高效率的存储模型解决纯MOLAP模型稀疏数据所造成的存储空间效率低下的问题.semi-MOLAP模型将一个多维查询划分为在不同数据集上独立的处理阶段,各阶段可以异步在地异构计算平台上执行,具有较好的分布式和流水并行处理能力,能够实现CPU平台和GPU平台的协同计算,提高平台的综合效率和性能.

本文提出了GPU memory resident为基础的多维数据分布模型,能够按平台的硬件性能配置性能最佳的数据集大小,当数据集超过最佳性能配置时,采用以最少数据传输为目标的基于水平分片的CPU和GPU协同计算模型来消除大数据在PCIe通道上的巨大数据传输延迟,最大化平台整体性能.本文的研究思想是从数据和计算特征出发,优化配置硬件计算资源,是一种软件(计算)订制硬件(计算资源)的策略.在未来的工作中,我们将研究面向给定硬件配置的可扩展数据量下的优化技术.

References:

[1] IBM informix warehouse accelerator-performance is everything. http://www.iiug.org/library/warehouse/technical/InformixWarehouseAcceleratorPaper.pdf

[2] Egham. Gartner says in-memory computing is racing towards mainstream adoption. 2013. http://www.gartner.com/newsroom/id/2405315

[3] Predicts 2014: In-memory computing will be adopted to deliver high-impact business value. 2013. http://www.gartner.com/doc/2629222/predicts--inmemory-computing-adopted

[4] Boncz PA, Kersten ML, Manegold S. Breaking the memory wall in MonetDB. Communications of the ACM, 2008,51(12):77−85. [doi:10.1145/1409360.1409380]

[5] http://www.top500.org/lists/2013/11/

[6] Kaczmarski K, Rudny T. MOLAP cube based on parallel scan algorithm. In: Eder J, Bielikova M, Tjoa AM, eds. Proc. of the Advances in Databases and Information Systems (ADBIS). LNCS 6906, Berlin, Heidelberg: Springer-Verlag, 2011. 125−138. [doi: 10.1007/978-3-642-23737-9_10]

[7] Lauer T, Datta A, Khadikov Z, Anselm C. Exploring graphics processing units as parallel coprocessors for online aggregation. In: Song IY, Ordonez C, eds. Proc. of ACM the 13th Int'l Workshop on Data Warehousing and OLAP (DOLAP) 2010. New York: ACM Press, 2010. 77−84. [doi: 10.1145/1871940.1871958]

[8] http://www.palo.net/index.php?id=12

[9] Malik M, Riha L, Shea C, El-Ghazawi TA. Task scheduling for GPU accelerated hybrid OLAP systems with multi-core support and text-to-integer translation. In: Proc. of the 26th IEEE Int'l Parallel and Distributed Processing Symp. Workshops (IPDPS) 2012. Washington: IEEE Computer Society, 2012. 1987−1996. [doi: 10.1109/IPDPSW.2012.259]

[10] Fang R, He BS, Lu M, Yang K, Govindaraju NK, Luo Q, Sander PV. GPUQP: Query co-processing using graphics processors. In: Chan CY, Ooi BC, Zhou AY, eds. Proc. of the SIGMOD Conf. 2007. New York: ACM Press, 2007. 1061−1063. [doi: 10.1145/1247480.1247606]

[11] He BS, Lu M, Yang K, Fang R, Govindaraju NK, Luo Q, Sander PV. Relational query coprocessing on graphics processors. ACM Trans. on Database System, 2009,34(4). [doi: 10.1145/1620585.1620588]

[12] He BS, Yang K, Fang R, Lu M, Govindaraju NK, Luo Q, Sander PV. Relational joins on graphics processors. In: Tsong J, Wang L, eds. Proc. of the SIGMOD 2008. New York: ACM Press, 2008. 511−524. [doi: 10.1145/1376616.1376670]

[13] Yuan Y, Lee RB, Zhang XD. The Yin and Yang of processing data warehousing queries on GPU devices. Proc. of the VLDB Endowment, 2013,6(10):817−828. [doi: 10.14778/2536206.2536210]

[14] Pirk H, Manegold S, Kersten M. Accelerating foreign-key joins using asymmetric memory channels. In: Bordawekar R, Lang CA, eds. Proc. of the Int'l Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures (ADMS). 2011. 27−35.

[15] Patel RA, Zhang Y, Mak J, Davidson A, Owens JD. Parallel lossless data compression on the GPU. In: Proc. of the Innovative Parallel Computing (InPar) 2012. Washington: IEEE, 2012. 1−9. [doi: 10.1109/InPar.2012.6339599]

[16] Zhang YS, Su MC, Zhou X, Wang S, Wang X. Keyword oriented bitmap join index for in-memory analytical processing. In: Wang JY, Xiong H, Ishikawa Y, Xu JL, Zhou JF, eds. Proc. of the Int'l Conf. on Web-Age Information Management (WAIM) 2013. LNCS 7923, Berlin, Heidelberg: Springer-Verlag, 2013. 405−416. [doi: 10.1007/978-3-642-38562-9_41]

[17] Zhang Y, Zhang YS, Su MC, Wang FZ, Chen H. HG-Bitmap join index: A hybrid GPU/CPU bitmap join index mechanism for OLAP. In: Huang ZS, Liu CF, He J, Huang GY, eds. Proc. of the Web Information Systems Engineering (WISE) 2013 Workshops BigWebData. LNCS 8182, Berlin, Heidelberg: Springer-Verlag, 2014. 23−26. [doi: 10.1007/978-3-642-54370-8_3]

[18] Zhen Z, Chen H, Zhang LY. Compilation and optimization of SQL query statements on column-oriented database. Computer Engineering, 2013,39(6):60−65 (in Chinese with English abstract).

[19] Walkowiak S, Wawruch K, Nowotka M, Ligowski L, Rudnicki W. Exploring utilisation of GPU for database applications. In: Sloot PMA, van Albada GD, Dongarra J, eds. Proc. of the Int'l Conf. on Computational Science (ICCS). Amsterdam: Elsevier-Procedia Computer Science, 2010. 505−513. [doi: 10.1016/j.procs.2010.04.054]

[20] Gregg C, Hazelwood K. Where is the data? Why you cannot debate CPU vs. GPU performance without the answer. In: Proc. of the IEEE Int'l Symp. on Performance Analysis of Systems and Software (ISPASS). Washington: IEEE Computer Society, 2011. 134−144. [doi: 10.1109/ISPASS.2011.5762730]

[21] Fang W, He B, Luo Q. Database compression on graphics processors. Proc. of the VLDB Endowment, 2010,3(1):670−680.

[22] Abadi DJ, Madden S, Ferreira M. Integrating compression and execution in column-oriented database systems. In: Chaudhuri S, Hristidis V, Polyzotis N, eds. Proc. of the SIGMOD 2006. New York: ACM Press, 2006. 671−682. [doi: 10.1145/1142473.1142548]

[23] Zhang NAF, Wu D, Stones DS, Wang G, Liu X, Liu J, Lin S. Efficient parallel lists intersection and index compression algorithms using graphics processing units. Proc. of the VLDB Endowment, 2011,4(8):470−481.

[24] Andrzejewski W, Wrembel R. GPU-WAH: Applying GPUs to compressing bitmap indexes with word aligned hybrid. In: Bringas PG, Hameurlain A, Quirchmayr G, eds. Proc. of the Database and Expert Systems Applications (DEXA) 2010. LNCS 6262, Berlin, Heidelberg: Springer-Verlag, 2010. 315−329. [doi: 10.1007/978-3-642-15251-1_26]

[25] http://developer.amd.com/wordpress/media/2013/06/1004_final.pdf

[26] He J, Lu M, He BS. Revisiting co-processing for Hash joins on the coupled CPU-GPU architecture. Proc. of the VLDB Endowment, 2013,6(10):889−900.

[27] Govindaraju NK, Lloyd B, Wang W, Lin MC, Manocha D. Fast computation of database operations using graphics processors. In: Weikum G, König AC, Deßloch S, eds. Proc. of the SIGMOD Conf. New York: ACM Press, 2004. 215−226. [doi: 10.1145/1007568.1007594]

[28] Govindaraju NK, Gray J, Kumar R, Manocha D. GPUTeraSort: High performance graphics co-processor sorting for large database management. In: Chaudhuri S, Hristidis V, Polyzotis N, eds. Proc. of the SIGMOD Conf. New York: ACM Press, 2006. 325−336. [doi: 10.1145/1142473.1142511]

[29] Bakkum P, Skadron K. Accelerating SQL database operations on a GPU with CUDA. In: Kaeli DR, Leeser M, eds. Proc. of the 3rd Workshop on General Purpose Processing on Graphics Processing Units. New York: ACM Press, 2010. 94−103. [doi: 10.1145/1735688.1735706]

[30] Wu HC, Diamos G, Cadambi S, Yalamanchili S. Kernel weaver: Automatically fusing database primitives for efficient GPU computation. In: Proc. of the 45th Annual IEEE/ACM Int'l Symp. on Microarchitecture (MICRO). Washington: IEEE Computer Society, 2012. 107−118. [doi: 10.1109/MICRO.2012.19]

[31] Breß S, Schallehn E, Geist I. Towards optimization of hybrid CPU/GPU query plans in database systems. In: Pechenizkiy M, Wojciechowski M, eds. New Trends in Databases and Information Systems, Workshop Proc. of the 16th East European Conf. on ADBIS. Berlin, Heidelberg: Springer-Verlag, 2012. 185. [doi: 10.1007/978-3-642-32518-2_3]

[32] Breß S. Why it is time for a HyPE: A hybrid query processing engine for efficient GPU coprocessing in DBMS. Proc. of the VLDB Endowment, 2013,6(12):1398−1403.

[33] Blanas S, Li Y, Patel JM. Design and evaluation of main memory hash join algorithms for multi-core cpus. In: Sellis TK, Miller RJ, Kementsietsidis A, Velegrakis Y, eds. Proc. of the SIGMOD. New York: ACM Press, 2011. 37−48. [doi: 10.1145/1989323.1989328]

[34] Balkesen C, Teubner J, Alonso G, Oszu MT. Main-Memory Hash joins on multi-core cpus: Tunning to the underlying hardware. In: Jensen CS, Jermaine CM, Zhou XF, eds. Proc. of the 29th Int'l Conf. on Data Engineering (ICDE). Washington: IEEE Computer Society, 2013. 362−373. [doi: 10.1109/ICDE.2013.6544839]

[35] Abadi DJ, Madden S, Hachem N. Column-Stores vs. row-stores: How different are they really? In: Tsong J, Wang L, eds. Proc. of the SIGMOD Conf. New York: ACM Press, 2008. 967−980. [doi: 10.1145/1376616.1376712]

[36] Rauhe H, Dees J, Sattler KU, Faerber F. Multi-Level parallel query execution framework for CPU and GPU. In: Catania B, Guerrini G, Pokorný J, eds. Proc. of the Advances in Databases and Information Systems, the 17th East European Conf. (ADBIS). LNCS 8133, Berlin, Heidelberg: Springer-Verlag, 2013. 330−343. [doi: 10.1007/978-3-642-40683-6_25]

[37] Heimel M, Saecker M, Pirk H, Manegold S, Markl V. Hardware-Oblivious parallelism for in-memory column-stores. Proc. of the VLDB Endowment, 2013,6(9):709−720.

附中文参考文献:

[18] 甄真,陈虎,张林亚.列数据库的 SQL 查询语句编译与优化.计算机工程,2013,39(6):60−65.

张宇(1977－),女,黑龙江绥化人,博士生,主要研究领域为 GPU,数据仓库,OLAP.

陈红(1965－),女,博士,教授,博士生导师,CCF 高级会员,主要研究领域为数据仓库,数据挖掘,传感器网络.

张延松(1973－),男,博士,副教授,主要研究领域为内存数据库,数据仓库,OLAP.

王珊(1944－),女,教授,博士生导师,CCF 会士,主要研究领域为高性能数据库,内存数据库,数据仓库.